Time series classification competition on Kaggle

SADI LIDIA*, DJAODOH OSSARA*, DIALLO AL HASSANE *, and AMOUWOTOR COMLAN *

This report presents our approach and results in a time series classification competition hosted on Kaggle. Our team's best performing model for step A : "Appliance Detection" was based on Time Series Forest, and the best performing model for step B : "Detection of appliance's activation period" was based on Bayesian analysis applied to the values of the time series data.

For step A, the contest consisted in predicting over a given period of time which of the 5 appliances were turned on. Our best score : 0.27 was obtained using the Time series Forest model. In the following lines we will explain what this approach consists of and the other approaches we tried with which we obtained a score between 0.13 and 0.23.

For step B, the competition involved predicting a binary classification label for a set of time series data points, and our approach involved extracting features from the time series data and using Bayesian analysis to classify the data into the two classes. Our model achieved an accuracy of 0.0236, outperforming several other popular classification algorithms such as Random Forest and Support Vector Machines which we tried. The report provides a detailed description of our approach, including the data pre-processing steps, feature extraction techniques, and the Bayesian analysis method used. We also provide insights into the strengths and limitations of our approach and suggestions for future work. Overall, our results demonstrate the potential effectiveness of Bayesian analysis for time series classification tasks.

Additional Key Words and Phrases: time series classification

ACM Reference Format:
Sadi Lidia, Djaodoh Ossara, Diallo Al Hassane, and Amouwotor Comlan . 2023. Time series classification competition on Kaggle. 1, 1 (April 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This project focuses on Time Series Classification (TSC), which is a crucial area of research with various applications in different domains. Like image classification or sentiment analysis in Natural Language Processing, the goal of TSC is to accurately classify a time series, which is a sequence of ordered points, according to a predefined class. A range of algorithms, including modern Machine Learning and Deep Learning methods, has been studied to achieve this objective.

Electricity suppliers have installed numerous smart meters worldwide in recent years, which record a large volume of time-stamped data series of electricity consumption, also known as load curves. These consumption time series offer valuable information such as detecting the presence of an appliance in a house or identifying the period in the data when the appliance is in an "ON" state. This project aims to propose a method to solve these two issues.

The project is divided into two parts: Firstly, to provide a solution for the classification problem of electricity consumption time series, which is to determine the presence or absence of various appliances. Secondly, to develop a method to detect the activation periods of these appliances in the time series, meaning to detect the time steps where a particular device is in an "ON" state. The ultimate goal of this project was to offer a comprehensive and effective approach to tackle the two objectives, utilizing advanced techniques in TSC, Machine Learning, and Deep Learning.

2 OUR APPROACH TO THE PROBLEMS

2.1 Part A

2.1.1 *Overview.*

The initial phase of this project involves training a classifier that can accurately identify whether one of the specified appliances is in an "ON" state based on the provided consumption time series data. The data has been preprocessed, and the associated labels indicating whether the appliances have been switched on at least once in the entire series have been provided. The training data set includes 10421 consumption time series of length 2160 and 10421 binary vectors of length 5 for the labels. The test set consists of 2488 consumption time series of length 2160 without labels. Participants are required to provide labels (0: not detected or 1: detected) or the probability of detection (floating value between 0 and 1) for the 5 appliances in the test set in a specific format. The solution may involve using binary classifiers, a multiclass classifier, or any other relevant approach. The evaluation metric is the mean average precision (MAP) calculated by finding the AP for each class and then averaging over a number of classes.

$$\text{MAP} = \frac{1}{|C|} \sum_{c \in C} \text{AP}_c$$

In this formula, C is the set of all classes, $|C|$ is the number of classes, and AP_c is the average precision for class c. The MAP value is the average of the AP values across all classes. To calculate the AP for each class, we first compute the precision and recall values for different threshold values. The precision is the fraction of true positives among the predicted positives, while the recall is the fraction of true positives among all the actual positives. The threshold values correspond to different confidence levels, and we compute the precision and recall values for each threshold value.

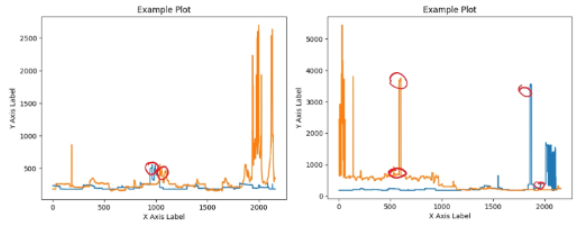
2.1.2 *Exploratory data analysis.*

First we retrieve the data from our dataset with pandas then we join on "Index" the trainInput with the trainLabel. Then we decided to filter our data and display them with matplotlib. We then retrieved all the data for which the label of the 5 appliances were 0, then the data for which only the dishwasher was on, and so on. Our main goal was to discover through the curves displayed by matplotlib, if there are repetition patterns between our different signals

*All authors contributed equally to this research.

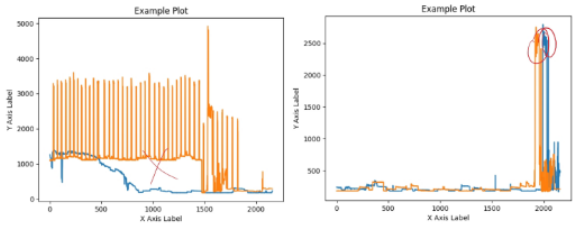
Authors' address: Sadi Lidia, lidia.sadi@etu.u-paris.fr; Djaodoh Ossara, elysee.djaodoh@etu.u-paris.fr; Diallo Al Hassane, al-hassane.diallo@etu.u-paris.fr; Amouwotor Comlan, comlan-todeale.amouwotor@etu.u-paris.fr.

XXXX-XXXX/2023/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



Dishwasher = 0, Kettle = 0, Microwave = 0, Tumble Dryer = 0, Washing Machine = 0

Fig. 1. comparative curve of two signals when neither appliance is on



Dishwasher = 0, Kettle = 0, Microwave = 0, Tumble Dryer = 0, Washing Machine = 1

Fig. 2. comparative curve of two signals when only Washing Machine is on

After comparison, we realized that the signals were not the same, even though we selected data from the same house when the same device is turned on at different times. This allows us to conclude that other devices turned on are interfering with our signal. We first thought of separating the signal into several parts, since it is a combination of signals from several devices, including the 5 we are trying to detect, but this will only be discussed in part b. So we tried different methods to classify our data.

2.1.3 *Models.*

For this part, we will briefly talk about the different models we have used and only focus on the one that has the best result in order to write a report that respects the page limit.

Support Vector Machine.

SVM stands for Support Vector Machine, which is a popular machine learning algorithm used for classification and regression analysis. It works by finding the best separating hyperplane between two classes of data, where the hyperplane is chosen to maximize the margin between the classes. The basic idea behind SVM is to map the input data into a high-dimensional feature space, where the classes become more separable. Then, the SVM algorithm tries to find the optimal hyperplane that separates the two classes with the largest possible margin. This hyperplane is defined by a subset of training data called support vectors. SVM can also handle non-linear classification problems by using a kernel function to map the input data into a higher-dimensional feature space, where the classes become separable. Examples of kernel functions include linear, polynomial, and radial basis function (RBF). SVM has been successfully applied in a wide range of applications, such as image classification, text classification, bio-informatics, and financial forecasting. Using this method, the result obtained is not as good because it gives us a score of 0.19, which is not a very good result. So, we are trying to use other methods to compare them to this one and see if we will get a better score.

Logistic Regression.

Logistic regression is a statistical method used for binary classification problems. It is a type of supervised learning algorithm that is used to model the relationship between a binary response variable and one or more predictor variables. The goal of logistic regression is to predict the probability of a binary outcome based on the values of the predictor variables.

The output of logistic regression is a probability value between 0 and 1. To make a binary classification decision, a threshold is set, and the probability value is compared to that threshold. If the probability is greater than the threshold, the prediction is made for the positive class; otherwise, it is made for the negative class.

Logistic regression can be used for both linear and nonlinear relationships between the predictor variables and the response variable. It is a popular method in various fields, including medicine, finance, and marketing, for predicting the probability of an event occurring based on a set of input variables. Using this method, the result obtained is not as good because it gives us a score of 0.23, which is not a very good result. So, we are trying to use other methods to compare them to this one and see if we will get a better score.

CNNs.

Convolutional Neural Networks (CNNs) [1] were originally designed for image recognition tasks, but they can also be used for time series classification. In this context, CNNs can be used to automatically extract relevant features from time series data and classify it into different categories.

```python
# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
  verbose, epochs, batch_size = 0, 10, 32
  n_timesteps, n_features, n_outputs = trainX.shape[1], 1, trainy.shape[1]
  model = Sequential()
  model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_timesteps,n_features)))
  model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
  model.add(Dropout(0.5))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Flatten())
  model.add(Dense(100, activation='relu'))
  model.add(Dense(n_outputs, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # fit network
  model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
  # evaluate model
  _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
  return accuracy
```

Fig. 3. CNN Architecture

We defined the CNN architecture: We designed a CNN architecture suitable for time series classification. We used 1D convolution, also specified the number of convolutions layers, filter size, clustering layers and the number of output nodes in the final layer.

We moved on to the training phase using labeled time series data. This involved feeding the data into the CNN, computing the loss function and optimizing the parameters using an optimization algorithm in our case "categorical$_c$*rossentropy*".

We evaluated the performance of the CNN using a validation set, the best score we obtained is 0.15.

Random Forest.

Random Forest is a supervised machine learning algorithm used for both classification and regression tasks. It is an ensemble learning method that uses multiple decision trees to make predictions. In a Random Forest, each decision tree is trained on a randomly selected subset of the input data and a randomly selected subset of features. During the training process, the Random Forest algorithm constructs a large number of decision trees, each based on a different set of randomly selected input data and features. Then, when making a prediction, the algorithm aggregates the outputs of all decision trees to produce the final prediction. This ensemble approach helps to improve the accuracy of the predictions and reduce overfitting to the training data.

We extracted the characteristics of the time series data. We used various time series features, such as mean, standard deviation, minimum, and maximum.

Once the features were extracted, we trained the Random Forest model on the labeled data set. The Random Forest algorithm works by constructing several decision trees, then combining their predictions to obtain a final classification. Each decision tree is trained on a subset of features and a subset of data to reduce overfitting.

After training the model, we were able to test it on the separate test set to evaluate its performance. The best score obtained was 0.19

While trying to dig a little further with the Random FOrest we came across Random Forest adapted for the Time series : Time series Forest

Time Series Forest.

The proposed method for classifying time series data is a tree-based ensemble approach called the time series forest (TSF). TSF evaluates splits using a combination of entropy gain and a distance measure, which is called the Entrance gain (entropy and distance). Experimental evidence indicates that the Entrance gain enhances the accuracy of TSF. TSF utilizes parallel computing techniques and randomly selects features at each tree node, resulting in a computational complexity that is linear in the time series length. The method also introduces the temporal importance curve to capture the temporal characteristics relevant for classification [2]. Simple features such as mean, standard deviation, and slope are used by TSF, which are computationally efficient and outperform powerful competitors like one-nearest-neighbor classifiers with dynamic time warping.

Explain the idea :

In the previous statement, we mentioned that Time Series Forest (TSF) is a machine learning algorithm, it is utilized for time series prediction and anomaly detection tasks. It is a variant of the random forest algorithm that is specifically tailored for time series data. The fundamental concept of a time series forest is to construct a collection of decision trees that can understand the prior behavior of a time series in order to make inferences about its future conduct. Each decision tree is taught on a distinct subset of the time series data, with varying starting points and durations of time series segments. During the training phase, each decision tree is given a random subset of the available features to split, which serves to prevent overfitting and enhance generalization. When making predictions, the TSF algorithm combines the predictions from all the individual decision trees in the forest to produce an overall prediction. One of the major advantages of the time series forest method is its capability to handle time series data with missing values and outliers more effectively than other conventional techniques. This is due to the fact that each decision tree is trained on a distinct subset of the data and can therefore adapt to different patterns and anomalies in the time series. In general, Time Series Forest is a flexible and robust machine learning algorithm that can be utilized for various time series prediction and anomaly detection tasks and has demonstrated competitiveness against other sophisticated techniques.

PseudoCode :

Algorithm : buildClassifierTSF (A list of n cases of length m, T = {X, y})

Parameters : the number of trees, r and the minimum subseries length, p.

Let F = <F1 … Fr> be the trees in the forest.

for i ← 1 to r do

  Let S be a list of n cases <s1, …, Sn> each with $3\sqrt{m}$ attributes

  for j ← 1 to $\lfloor\sqrt{m}\rfloor$ do

    a = rand(1, m-p)

    b = rand(s+p, m)

    for k ← 1 to n do

      $s_k, 3(j-1)+1 = mean\ (x_k, a, b)$

      $s_k, 3(j-1)+2 = standardDeviation\ (x_k, a, b)$

      $s_k, 3(j-1)+3 = slope\ (x_k, a, b)$

    F_i.buildClassifier({S, y})

Fig. 4. TSF Algorithm

Experiments

Hardware :

For the hardware part not having a computer we opted for google colabs in free version

Datasets :

This project utilized a dataset containing multiple time series of electricity consumption data from 9 houses, with each house's data sampled at a 10-second frequency. The dataset includes both total consumption power and appliance level power for each house, allowing for identification of when a specific appliance was switched on. To facilitate analysis, the consumption curves were preprocessed and split into 6-hour periods, resulting in multiple univariate time series of length 2160. For this project, 5 appliances were selected for analysis: Washing Machine, Dishwasher, Tumble Dryer, Microwave,

and Kettle. The dataset for training includes 10421 time series of energy consumption, each having a length of 2160. Additionally, it comprises 10421 binary vectors of length 5, indicating whether the 5 specified appliances were in the "ON" state at any point during the entire series. These series and vectors can be utilized to develop and assess models. In contrast, the test set comprises 2488 consumption time series of length 2160, but it lacks labels.

Describe :

First of all, we have collected our data with the pandas tool: the train and the labels We then join on column "Index". We divided the dataset in two parts 0.7 for training and 0.3 for testing. We deleted the columns Index, House¡*dwhichwillnotbeusedinthetraining*.

Results:

We scored 0.27 which is still our best score. We thought that our results were so low because the data is an aggregation of signals

2.2 Part B

2.2.1 Overview. All our previous models were not giving us high enough scores. We concluded that our models were parasitized by the presence of other signals so we tried to find new methods to disaggregate our signals. But before that, the goal of part b is to detect the activation periods of our 5 appliances, which corresponds to a classification on 2160 columns * 5 appliances * 10421 rows. We realize that our research will take us some time. We tried all our previous models in part A on our data which took 9 days of processing to give us strangely that either SVM, TSF, CNN, Random Forest a score of 0.00846

During the research done at the same time we came across articles about NILM (Non-Intrusive Load Monitoring) which is an energy disaggregation system. Especially we came across the article "Deep Learning Based Energy Disaggregation and On/Off Detection of Household Appliances".

2.2.2 WaveNets.

The process of Energy disaggregation, also known as Non-Intrusive Load Monitoring, is aimed at separating the energy usage of individual devices from the overall energy consumption measured by a mains power meter, such as that of a whole household. The identification of individual appliances' energy consumption can be advantageous in numerous scenarios, for instance, by providing device-specific feedback to users to help them comprehend their energy usage and ultimately reduce it. In recent times, several neural network models, including convolutional and recurrent neural networks, have been studied to tackle the energy disaggregation issue, thanks to the availability of large-scale energy consumption datasets. Neural network models can learn complex patterns from significant amounts of data and have been observed to outperform traditional machine learning approaches such as hidden Markov models' variations. However, the current neural network techniques for energy disaggregation are either computationally expensive or insufficient in handling long-term dependencies[3].

In the article we read, we explore the application of WaveNet models, which have recently been developed, for energy disaggregation tasks.

WaveNet :

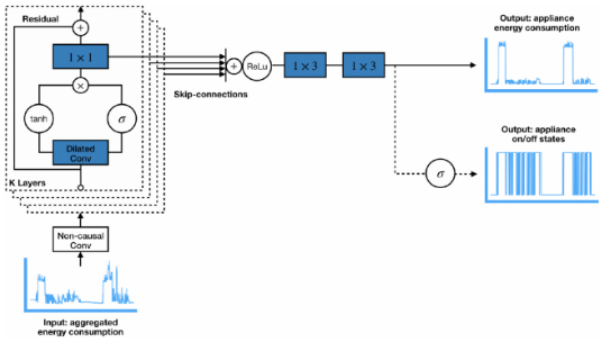


Fig. 5. Architecture of WaveNets for energy disaggregation and on/off detection.

Figure 5 displays the WaveNet architecture utilized for energy disaggregation and on/off detection. The model accepts a series of aggregate energy readings as inputs, which are initially processed by a non-causal convolutional layer, followed by a sequence of residual blocks featuring dilated layers embedded within them. The output of each residual block is fed as input to the subsequent block. Inside each residual block, an input is subjected to a dilated convolutional layer, which is followed by a Tanh layer and a Sigmoid layer. The outputs of these two layers are multiplied and then passed through a 1×1 convolutional layer. The sum of the skip output of all residual blocks is then computed and fed to two 3 × 1 convolutional layers to obtain the final appliance energy consumption output, which is useful in energy disaggregation. To perform on/off detection, a Sigmoid layer is added to generate the final appliance on/off state predictions.

Experiments (Fig 6) :

We tried to set up the architecture but in front of its complexity, on all our lack of knowledge and especially the lack of time we had to stop for a method me complex for us. So we turned to Bayesian analysis techniques

2.2.3 Exploratory data analysis. In order to get a better grasp of the content of the data we did the following analysis:

- perform a statistical rendering of the various combinations of data points
- visualize the data and

Our code for Part B contains multiple visualizations of time-series data for exploring a dataset of the time-series. The code first defines a list of colors, sets values for line, inf, and sup, and then creates six subplots to display various ranges of the time series data. This allows us to zoom into the data anywhere to observe what happens in there.

The first plot shows the time series data for a 6-hour window using the matplotlib's plot function, where the *c* parameter sets the color of the line to blue. This plot provides an overall view of the

time-series data for the specified window, with a label indicating the line number of the data.

The second plot shows the differential values of the time series data using the $differential\_series$ function with a threshold value of 300. The ax.plot function plots the differential values with a brown color line and a label 'differential' indicating the values represent the changes in the time-series data.

The next four plots show the activation indicators of different home appliances such as washing machine, dish washer, tumble dryer, microwave, and kettle for the specified time-series data window. The ax.plot function plots the activation indicator data of each appliance with a different color and a label indicating the appliance name. The washing machine data is plotted with a red color line, the dish washer data with a yellow color line, the tumble dryer data with a black color line, the microwave data with a gray color line, and the kettle data with a green color line.

Overall, these visualizations help to analyze the time-series data of different appliances and provide insights into the patterns and trends in the data.

After sorting the entries with respect to the number of data points associated with each combination, we obtain the following graph:

2.2.4 Pre-processing.

2.2.5 Models. The proposed method for classifying time series data is a tree-based ensemble approach called the time series forest (TSF). TSF evaluates splits using a combination of entropy gain and a distance measure, which is called the Entrance gain (entropy and distance). Experimental evidence indicates that the Entrance gain enhances the accuracy of TSF. TSF utilizes parallel computing techniques and randomly selects features at each tree node, resulting in a computational complexity that is linear in the time series length. The method also introduces the temporal importance curve to capture the temporal characteristics relevant for classification [1]. Simple features such as mean, standard deviation, and slope are used by TSF, which are computationally efficient and outperform powerful competitors like one-nearest-neighbor classifiers with dynamic time warping.

In the previous statement, we mentioned that Time Series Forest (TSF) is a machine learning algorithm, it is utilized for time series prediction and anomaly detection tasks. It is a variant of the random forest algorithm that is specifically tailored for time series data. The fundamental concept of a time series forest is to construct a collection of decision trees that can understand the prior behavior of a time series in order to make inferences about its future conduct. Each decision tree is taught on a distinct subset of the time series data, with varying starting points and durations of time series segments. During the training phase, each decision tree is given a random subset of the available features to split, which serves to prevent overfitting and enhance generalization. When making predictions, the TSF algorithm combines the predictions from all the individual decision trees in the forest to produce an overall prediction. One of the major advantages of the time series forest method is its capability to handle time series data with missing values and outliers more effectively than other conventional techniques. This is due to the fact that each decision tree is trained on a distinct subset of the data and can therefore adapt to different patterns and anomalies in the time series. In general, Time Series Forest is a flexible and robust machine learning algorithm that can be utilized for various time series prediction and anomaly detection tasks and has demonstrated competitiveness against other sophisticated techniques.

Combination of appliances	Number of data points
no machine	21051303
tumble	70749
kettle	21806
tumble+kettle	244
micro	77969
micro+tumble	706
micro+kettle	144
micro+tumble+kettle	7
dish	589427
dish+tumble	1152
dish+kettle	284
dish+tumble+kettle	20
dish+micro	3329
dish+micro+tumble	46
dish+micro+kettle	8
dish+micro+tumble+kettle	0.0
wash	565892
wash+tumble	1690
wash+kettle	734
wash+tumble+kettle	24
wash+micro	21622
wash+micro+tumble	311
wash+micro+kettle	88
wash+micro+tumble+kettle	5
wash+dish	49268
wash+dish+tumble	117
wash+dish+kettle	23
wash+dish+tumble+kettle	0
wash+dish+micro	1125
wash+dish+micro+tumble	44
wash+dish+micro+kettle	4
wash+dish+micro+tumble+kettle	0

2.3 Bayesian approach

The idea of used in this Part B is to compute for each of the possible combinations of appliances, the frequencies of the different values electricity load at during the period of activation of those appliances.

```python
class NonCasualDilatedConv1D(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, dilation, padding=1):
        super().__init__()
        self.conv1D = nn.Conv1d(in_channels, out_channels, kernel_size, dilation=dilation, bias=False, padding='same')
        self.ignoreOutIndex = (kernel_size - 1) * dilation

    def forward(self, x):
        return self.conv1D(x)[..., :-self.ignoreOutIndex]
```

```python
class ResBlock(nn.Module):
    def __init__(self, res_channels, skip_channels, kernel_size, dilation):
        super().__init__()
        self.casualDilatedConv1D = CasualDilatedConv1D(res_channels, res_channels, kernel_size, dilation=dilation)
        self.resConv1D = nn.Conv1d(res_channels, res_channels, kernel_size=1)
        self.skipConv1D = nn.Conv1d(res_channels, skip_channels, kernel_size=1)
        self.tanh = nn.Tanh()
        self.sigmoid = nn.Sigmoid()

    def forward(self, inputX, skipSize):
        x = self.casualDilatedConv1D(inputX)
        x1 = self.tanh(x)
        x2 = self.sigmoid(x)
        x = x1 * x2
        resOutput = self.resConv1D(x)
        resOutput = resOutput + inputX[..., -resOutput.size(2):]
        skipOutput = self.skipConv1D(x)
        skipOutput = skipOutput[..., -skipSize:]
        return resOutput, skipOutput
```

```python
class StackOfResBlocks(nn.Module):
    def __init__(self, stack_size, layer_size, res_channels, skip_channels, kernel_size):
        super().__init__()
        buildDilationFunc = np.vectorize(self.buildDilation)
        dilations = buildDilationFunc(stack_size, layer_size)
        self.resBlocks = []
        for s,dilationPerStack in enumerate(dilations):
            for l,dilation in enumerate(dilationPerStack):
                resBlock=ResBlock(res_channels, skip_channels, kernel_size, dilation)
                self.add_module(f'resBlock_{s}_{l}', resBlock)
                self.resBlocks.append(resBlock)

    def buildDilation(self, stack_size, layer_size):
        dilationsForAllStacks = []
        for stack in range(stack_size):
            dilations = []
            for layer in range(layer_size):
                dilations.append(2 ** layer)
            dilationsForAllStacks.append(dilations)
        return dilationsForAllStacks

    def forward(self, x, skipSize):
        resOutput = x
        skipOutputs = []
        for resBlock in self.resBlocks:
            resOutput, skipOutput = resBlock(resOutput, skipSize)
            skipOutputs.append(skipOutput)
        return resOutput, torch.stack(skipOutputs)
```

```python
class WaveNet(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stack_size, layer_size):
        super().__init__()
        self.stack_size = stack_size
        self.layer_size = layer_size
        self.kernel_size = kernel_size
        self.casualConv1D = CasualDilatedConv1D(in_channels, in_channels, kernel_size, dilation=1)
        self.stackResBlock = StackOfResBlocks(self.stack_size, self.layer_size, in_channels, out_channels, kernel_size)
        self.denseLayer = DenseLayer(out_channels)


    def calculateReceptiveField(self):
        return np.sum([(self.kernel_size - 1) * (2 ** l) for l in range(self.layer_size)] * self.stack_size)

    def calculateOutputSize(self, x):
        return int(x.size(2)) - self.calculateReceptiveField()

    def forward(self, x):
        x = self.casualConv1D(x)
        skipSize = self.calculateOutputSize(x)
        _, skipConnections = self.stackResBlock(x, skipSize)
        dense=self.denseLayer(skipConnections)
        return dense
```

```python
class WaveNet(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stack_size, layer_size):
        super().__init__()
        self.stack_size = stack_size
        self.layer_size = layer_size
        self.kernel_size = kernel_size
        self.casualConv1D = CasualDilatedConv1D(in_channels, in_channels, kernel_size, dilation=1)
        self.stackResBlock = StackOfResBlocks(self.stack_size, self.layer_size, in_channels, out_channels, kernel_size)
        self.denseLayer = DenseLayer(out_channels)


    def calculateReceptiveField(self):
        return np.sum([(self.kernel_size - 1) * (2 ** l) for l in range(self.layer_size)] * self.stack_size)

    def calculateOutputSize(self, x):
        return int(x.size(2)) - self.calculateReceptiveField()

    def forward(self, x):
        x = self.casualConv1D(x)
        skipSize = self.calculateOutputSize(x)
        _, skipConnections = self.stackResBlock(x, skipSize)
        dense=self.denseLayer(skipConnections)
        return dense
```

Fig. 6. Implementation of WaveNets Architecture.

Combinations of appliances	3 most frequent load values
no machine	[(180, 219410), (159, 206503)]
tumble	[(6000, 685), (2993, 119)]
kettle	[(6000, 307), (1511, 62)]
tumble+kettle	[(6000, 73), (4444, 6)]
micro.	[(6000, 592), (401, 129)]
micro.+tumble	[(6000, 184), (4834, 8)]
micro.+kettle	[(6000, 87), (4758, 9)]
micro.+tumble+kettle	[(6000, 7), (4154, 2)]
dish.	[(2169, 242689), (2729, 4392)]
dish.+tumble	[(6000, 212), (5258, 9)]
dish.+kettle	[(6000, 39), (3950, 5)]
dish.+tumble+kettle	[(6000, 21), (5943, 1)]
dish.+micro.	[(6000, 197), (5213, 14)]
dish.+micro.+tumble	[(6000, 24), (5120, 4)]
dish.+micro.+kettle	[(6000, 9), (5908, 1)]
dish.+micro.+tumble+kettle	[(0, 0), (1, 0)]
wash.	[(6000, 3696), (602, 813)]
wash.+tumble	[(6000, 134), (3650, 9)]
wash.+kettle	[(6000, 113), (2126, 9)]
wash.+tumble+kettle	[(6000, 24), (4795, 2)]
wash.+micro.	[(6000, 854), (5037, 38)]
wash.+micro.+tumble	[(6000, 222), (5524, 8)]
wash.+micro.+kettle	[(6000, 70), (4312, 3)]
wash.+micro.+tumble+kettle	[(6000, 6), (4768, 1)]
wash.+dish.	[(6000, 853), (918, 93)]
wash.+dish.+tumble	[(6000, 106), (5017, 3)]
wash.+dish.+kettle	[(6000, 14), (3763, 2)]
wash.+dish.+tumble+kettle	[(5876, 1), (0, 0)]
wash.+dish.+micro.	[(6000, 577), (5266, 8)]
wash.+dish.+micro.+tumble	[(6000, 41), (5149, 3)]
wash.+dish.+micro.+kettle	[(6000, 5), (4288, 1)]
wash.+dish.+micro.+tumble+kettle	[(6000, 1), (0, 0)]

The previous table presents for each combination of appliances, the 3 most frequent for various load values. Based on this analysis, we developed a simple Bayesian model that predicts for each value in test time series the most likely combination of appliances associated with that value. This approach yielded our best result during this second part of the competition.

3 CONCLUSION

In this report we presented an overview of the methods and experimentations that we used in our attempt to tackle the challenges of the competition. Our results, although not the best we could have obtained given more experience and time, have been obtained through careful and diligent work. This competition has allowed us to get a first taste of the work a data scientist. This has been an eye-opening experience.